Solutions - Chapter 11

11-1: City, Country

Write a function that accepts two parameters: a city name and a country name. The function should return a single string of the form *City, Country*, such as Santiago, Chile. Store the function in a module called *city\_functions.py*.

Create a file called *test\_cities.py* that tests the function you just wrote (remember that you need to import unittest and the function you want to test). Write a method called test\_city\_country() to verify that calling your function with values such as santiago and chile results in the correct string. Run *test\_cities.py*, and make sure test\_city\_country() passes.

*city\_functions.py:*

"""A collection of functions for working with cities."""

**def** **city\_country**(city, country):

"""Return a string like 'Santiago, Chile'."""

**return**(city**.**title() **+** ", " **+** country**.**title())

***Note:****This is the function we wrote in Exercise 8-6.*

*test\_cities.py:*

import unittest

from city\_functions import city\_country

**class** **CitiesTestCase**(unittest**.**TestCase):

"""Tests for 'city\_functions.py'."""

**def** **test\_city\_country**(self):

"""Does a simple city and country pair work?"""

santiago\_chile **=** city\_country('santiago', 'chile')

self**.**assertEqual(santiago\_chile, 'Santiago, Chile')

unittest**.**main()

Output:

.

----------------------------------------------------------------------

Ran 1 test in 0.000s

OK

11-2: Population

Modify your function so it requires a third parameter, population. It should now return a single string of the form City, Country - population xxx, such as Santiago, Chile - population 5000000. Run*test\_cities.py* again. Make sure test\_city\_country() fails this time.

Modify the function so the population parameter is optional. Run *test\_cities.py* again, and make sure test\_city\_country() passes again.

Write a second test called test\_city\_country\_population() that verifies you can call your function with the values 'santiago', 'chile', and 'population=5000000'. Run *test\_cities.py* again, and make sure this new test passes.

Modified *city\_functions.py*, with required population parameter:

"""A collection of functions for working with cities."""

**def** **city\_country**(city, country, population):

"""Return a string like 'Santiago, Chile - population 5000000'."""

output\_string **=** city**.**title() **+** ", " **+** country**.**title()

output\_string **+=** ' - population ' **+** str(population)

**return** output\_string

Output from running *test\_cities.py:*

E

======================================================================

ERROR: test\_city\_country (\_\_main\_\_.CitiesTestCase)

Does a simple city and country pair work?

----------------------------------------------------------------------

Traceback (most recent call last):

File "pcc\solutions\test\_cities.py", line 10, in test\_city\_country

santiago\_chile = city\_country('santiago', 'chile')

TypeError: city\_country() missing 1 required positional argument: 'population'

----------------------------------------------------------------------

Ran 1 test in 0.000s

FAILED (errors=1)

Modified *city\_functions.py*, with optional population parameter:

"""A collection of functions for working with cities."""

**def** **city\_country**(city, country, population**=**0):

"""Return a string representing a city-country pair."""

output\_string **=** city**.**title() **+** ", " **+** country**.**title()

**if** population:

output\_string **+=** ' - population ' **+** str(population)

**return** output\_string

Output of running *test\_cities.py:*

.

----------------------------------------------------------------------

Ran 1 test in 0.001s

OK

Modified *test\_cities.py:*

import unittest

from city\_functions import city\_country

**class** **CitiesTestCase**(unittest**.**TestCase):

"""Tests for 'city\_functions.py'."""

**def** **test\_city\_country**(self):

"""Does a simple city and country pair work?"""

santiago\_chile **=** city\_country('santiago', 'chile')

self**.**assertEqual(santiago\_chile, 'Santiago, Chile')

**def** **test\_city\_country\_population**(self):

"""Can I include a population argument?"""

santiago\_chile **=** city\_country('santiago', 'chile', population**=**5000000)

self**.**assertEqual(santiago\_chile, 'Santiago, Chile - population 5000000')

unittest**.**main()

Output:

..

----------------------------------------------------------------------

Ran 2 tests in 0.000s

OK

11-3: Employee

Write a class called Employee. The \_\_init\_\_() method should take in a first name, a last name, and an annual salary, and store each of these as attributes. Write a method called give\_raise() that adds $5000 to the annual salary by default but also accepts a different raise amount.

Write a test case for Employee. Write two test methods, test\_give\_default\_raise() andtest\_give\_custom\_raise(). Use the setUp() method so you don’t have to create a new employee instance in each test method. Run your test case, and make sure both tests pass.

*employee.py:*

**class** **Employee**():

"""A class to represent an employee."""

**def** **\_\_init\_\_**(self, f\_name, l\_name, salary):

"""Initialize the employee."""

self**.**first **=** f\_name**.**title()

self**.**last **=** l\_name**.**title()

self**.**salary **=** salary

**def** **give\_raise**(self, amount**=**5000):

"""Give the employee a raise."""

self**.**salary **+=** amount

*test\_employee.py:*

import unittest

from employee import Employee

**class** **TestEmployee**(unittest**.**TestCase):

"""Tests for the module employee."""

**def** **setUp**(self):

"""Make an employee to use in tests."""

self**.**eric **=** Employee('eric', 'matthes', 65000)

**def** **test\_give\_default\_raise**(self):

"""Test that a default raise works correctly."""

self**.**eric**.**give\_raise()

self**.**assertEqual(self**.**eric**.**salary, 70000)

**def** **test\_give\_custom\_raise**(self):

"""Test that a custom raise works correctly."""

self**.**eric**.**give\_raise(10000)

self**.**assertEqual(self**.**eric**.**salary, 75000)

unittest**.**main()

Output:

..

----------------------------------------------------------------------

Ran 2 tests in 0.000s

OK